The functional partitioning of memory – its application to interactive computing

J. N. Sturman

Bell Telephone Laboratories Incorporated, Murray Hill, New Jersey, USA

The objective of the effort described in this paper has been the design of computer software to reduce memory occupancy of interactive programs. This effort has resulted in the development of a batch-compatible interactive system which has been operational at Bell Telephone Laboratories for over a year. In that time a small, closely monitored community of users has been established. Some details of software design are presented and measurements of interactive system performance are examined.

(Received September 1970)

This paper is a design description of a software system which adds interactive computing facilities to a multiprogramming batch-oriented monitor system. In the pages that follow, the design objectives will be stated, some details of software will be presented, and measurements of interactive system performance will be discussed. However, this paper is not intended to serve as a treatise on the design of interactive systems. Rather, it is a description of the technique that was employed, the functional partitioning of memory; the resultant interactive system being an example of specific application.

The objective of the effort described in this paper has been the design of computer software to reduce memory occupancy of programs likely to be resident for long durations. In particular, programs requiring human interaction are characterised by their demand for relatively little computing power over long periods of time. If one expects a computing installation to provide economical service to many such interactive programs, an effective strategy of core memory management is mandatory.

The GE 635 computer operating under GECOS III served as the available host computer and associated software environment. The unavailability of paging hardware in this machine precluded any further consideration of this approach to providing effective memory management.

A second proposed solution, broadly classified as the 'super system' approach, was taken by the General Electric Company in their design of the TSS subsystem for GECOS III (see Reference). It is the task of the time-sharing monitor to perform the functions of memory management and dispatching among its many users, to enforce some discipline over file system and resource allocation, to intercept and interpret system level commands, etc. This time-sharing module, in turn, appears to the GECOS monitor as a single job to which the supervisor allocates a fraction of *its* total system resources. The allocation by GECOS of peripherals, core memory and processor time is in a sense reperformed and suballocated by the time-sharing module. The replication of tasks clearly within the domain of the system monitor and the difficulties in isolating a system-within-a-system seem to have led to time-sharing/batch-operation system incompatibilities.

A third approach, and that taken by this author, is to provide multiple copies of a dedicated one-per-user interactive facility with low memory residence requirements. These small single user systems rely heavily upon the GECOS supervisor for necessary support in the multiprogramming environment.

The following paper describes the design of a GERTS User's Terminal System* in which this third approach was taken. This interactive system, henceforth referred to by the acronym of 'GUTS', has been operational at Bell Telephone Laboratories for over a year and in that time has established a small, closely monitored community of users.

*GERTS is an acronym for the GEneral Remote Terminal Supervisor which controls the remote line multiplex and interface discipline for GECOS.

GUTS is compatible with the batch environment. It provides the user with interactive editing facilities, along with many of the software tools available to batch users. These include access to the batch (unmodified) FORTRAN, ALGOL and COBOL compilers as well as the GMAP assembler. Other facilities include access to the standard loader, the use of SNOBOL, a string manipulation language, access to manufacturer distributed utility programs as well as easy access to a host of subroutines and subsystems available, and unique to, the user installation. In short, the GUTS user has complete batch compatibility.

Design objectives

Compatibility

As previously indicated, one of the primary design objectives has been compatibility with batch mode computing, GUTS uses the same character set, it has access to bulk system output printer and card punch facilities, it uses the same libraries, it can access the same file system, and it reacts intelligently when encountering binary object and binary data card images in a file of text or control cards.

An important side effect of software compatibility is the ease of system maintenance. The module loaded upon the issuance of a 'FORTRAN' command is not just a *compatible* FORTRAN compiler it is *the same* FORTRAN compiler as used in the batch environment. Similarly, all compilers, assemblers, local and manufacturer supplied utility subroutines are identical. GUTS keeps no private copies of systems or facilities for its own use.

Minimisation of memory residence

GUTS operation can be viewed as providing the user with two different computing environments. One is highly interactive, places little load on the processor, and is written to require 2048 (36 bit) words of core memory per user*. The second computing environment is one in which a significant amount of computing is performed. It is not interactive and requests core as needed from the multiprogramming system monitor. FORTRAN, ALGOL, GMAP and user jobs are run in this

*Present operation requires 5120 words of memory as follows:

1. 2048 words for the GUTS system.
2. 1024 words (slave service area) required by the monitor system for each resident program in which is stored critical system tables and code directly attributable to the user.
3. 2048 words for (temporary) software which measures GUTS operational performance.

environment. The key to the facility provided by GUTS is its ability to alternate between these two modes of operation in order to minimise core memory residence. Thus, when in interactive mode, obeying relatively trivial editing, updating and file system commands for the user, GUTS core occupancy is low. When executing a user's program, compiling, or assembling, considerably greater memory occupancy is required but for a relatively short period of time. It is this feature of dynamic memory occupancy that distinguishes GUTS from other approaches to interactive batch-compatible computing under GECOS.

Facility before fail-safe

There occur many instances in design of a software system where a go, no-go decision must be made in permitting a user to issue a command which may get him into some difficulty. In some cases the decision is an easy one, since the request is entirely unreasonable. However, there are cases where command validity is not easily determined and arbitrary denial would place an unnecessary restriction upon the user. On the other hand, trying to execute an invalid command may result in a fatal termination of the user. By providing separate duplicate systems for each user, fatal errors result in inconvenience only to the offending user and thus a significant measure of flexibility as well as user isolation is provided. The same philosophy applies to experimental versions of the system in various stages of development before being made publicly available.

User feedback

Another important design objective was the establishment of a community of users which could provide useful suggestions relevant to future implementations. It was considered important to keep the user group small, separating the honest users from the casual observers and weighting a user's suggestions and criticisms by his level of computing sophistication. A mechanism was established to log a user's access to the system, noting his date of last access and frequency of use. The resulting data was used to prune the distribution lists of successive updates of user's manuals. Detailed user characteristics were also kept dealing with user response time, system performance, the nature of the user commands, the user's memory residence, his demand upon secondary storage devices, etc. This data yielded information relevant to the characteristics of the user as well as characteristics of the system performance.

System organisation

GUTS is an interactive, one-user system, designed to be com-

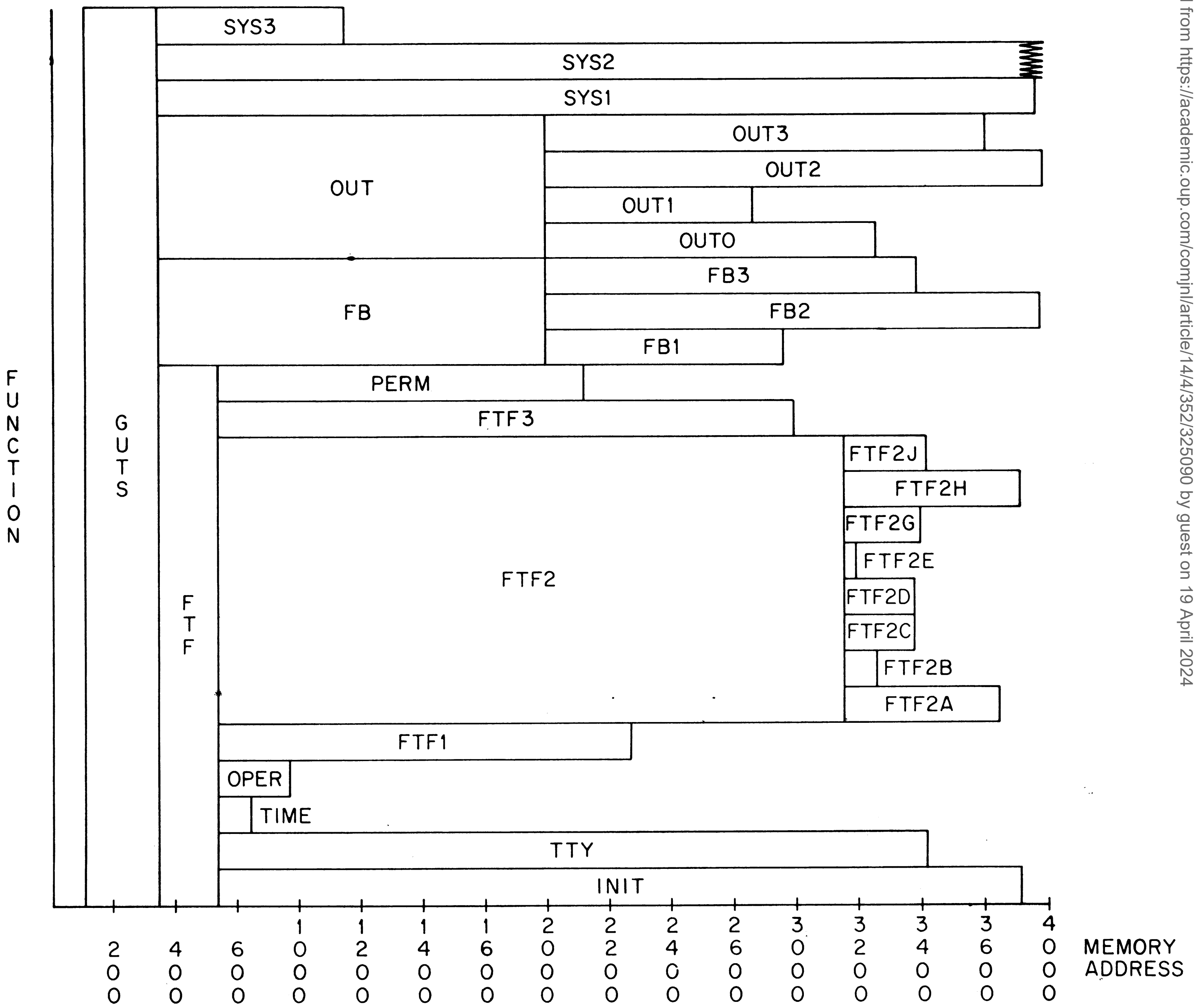


Fig. 1

patible with batch GECOS, and having modest memory requirements when in interactive mode. A new user, logging into GUTS, brings a fresh copy of system object code into core which services his needs. Thus it is possible, and indeed highly likely for several simultaneous interactive users to be working in GUTS, each user dealing with his own private copy of the system.

Interactive modules

The problem of how to reduce memory residence of interactive users was a critical aspect in the design of GUTS. The need for interactive editing was the primary factor determining the size of the overall system.

Since GECOS charges the slave programs with the responsibility of allocating input-output buffers and loading most of the I/O software required to deal with files, it soon became apparent that the core residence requirements of even a simple contiguous editor would be uneconomical. By partitioning the subtasks required by the editor it was found that significant saving of core memory would result. For example, within the 2048 words of memory allotted to each user of GUTS, one could easily accommodate the code required to open two editing files with their associated file control blocks and buffers. However, once opened for I/O, there was not sufficient room in memory to accommodate the code required for reading or writing the file. Similarly, within 2K (1K = 1024 words), file I/O could be performed with sufficient available space to perform a single editing function such as input from a teletype, output to a teletype, context searching or binary deck manipulation. The function of file closing required still another separate module.

In addition to the modules which performed the above tasks, other modules performed functions of command decoding, file system access and master console operator communication. **Fig. 1** is a pictorial representation of the system memory map, the abscissa representing the octal locations in core (note $4000_8 = 2048_{10}$) and the ordinate representing function. For example, a horizontal line drawn through modules GUTS, FTF, FTF2 and FTF2C represents core configuration when editing in input mode, i.e. when constructing a file from a teletype. **Table 1** is a tabulation of the module names and their associated functions.

In some sense, nothing new is being presented, since the structuring of large programs into links or overlays is an established technique for reducing memory residence. However, it is significant to note the degree to which the task has been segmented and the interrelationship between the modules. On one hand, a program structured in links or overlays usually takes the form of a tree with a main link or controlling overlay calling subordinate modules into core as needed. GUTS features no such main-subordinate relationship between its modules, and although the module labeled 'GUTS' appears to be the controlling vertex of the system, it contains almost no executable object code but rather is devoted to the storage of an editing buffer, a file control block and other data of global significance to the other modules.

An examination of the GUTS system code would reveal that control resides in the modules which form the terminal nodes of the tree (e.g. FTF2A, FTF2B, FTF2C, etc.) while internal nodes such as FTF2 always contain data and service subroutines. From a user's point of view, TTY is the controlling module since it performs the functions of command decoding and teletype communication when at command level. This module is resident for 45% of a typical session consuming 7% of GUTS processor time.

When in editing mode, modules GUTS and FTF are resident, each containing a file buffer and other pertinent data for an editing scratch file. Modules FTF1 and FTF3 perform the respective operations of opening and closing these files while FTF2 reads and writes the files. The specific editing function determines which one of the utility modules, FTF2A, FTF2B, FTF2C, FTF2D, etc. is to be loaded for functions of printing a file on the teletype, file positioning, input from a teletype, generation of punched paper tape, etc.

Table 1

MODULE NAME	FUNCTION
GUTS	Contains one buffer and associated control parameters for one of the internal editing files (usually the output file), static GUTS variables and reserved locations for inter-module communication.
FTF	Contains the second editing (input) buffer and file control parameters.
INIT	Initialisation code and error recovery procedures.
TTY	Command level decoding, teletype communication and execution of simple commands.
TIME	Informs the user of elapsed processor time and time of day since initial log on.
OPER	Communication between the interactive user and computer operators.
FTF1	File opening code for editing files of modules GUTS and FTF.
FTF2	Input-output code for reading and writing editing files.
FTF2A	Contains Teletype code for listing an editing file.
FTF2B	Positions editing files.
FTF2C	Contains Teletype input code for building a file from a Teletype (input mode).
FTF2D	Used for generating punched paper tape escapes on teletype.
FTF2E	GUTS interface for sending text to on-line cathode ray tube display.
FTF2G	Converts compressed binary text card images to hollerith card images.
FTF2H	Interprets loader control cards on binary object decks (used for scanning libraries of object code).
FTF2J	Contains ceze for context scanning.
FTF3	File closing code for editing files of modules GUTS and FTF.
PERM	Controls access to semi-permanent secondary storage files.
FB	Contains an input buffer for reading standard system files.
FB1	Opens file buffers contained in modules GUTS and FB.
FB2	Reads and positions file of module FB, copies records from FB to GUTS.
FB3	Closes file buffers contained in modules GUTS and FB.
OUT	Contains an output buffer for writing standard system files.
OUTO	Temporarily converts the output file of module GUTS to an input file.
OUT1	Opens files of modules OUT and GUTS.
OUT2	Copies a file from GUTS to OUT.
OUT3	Closes file of module OUT.
SYS1	Grows core for non-interactive mode.
SYS2	Copies diagnostics or program output (if any) and returns GUTS to appropriate size.
SYS3	Protected mode supervisor.

Modules FB1, FB2 and FB3 are used to manipulate files in standard system format and to copy one or more external files into the editor scratch files. Modules prefixed by the letters 'OUT' move data from the editing files to standard system files or to system output for bulk printing or punching. Module OPER is loaded when the user wishes to communicate with the computer operator stationed at the main console or at the tape mounting station. The PERM module is used to gain access to the file system.

Non-interactive Modules

The modules prefixed by the characters 'SYS' serve as an interface between the low memory resident, interactive facility and the high core resident, non-interactive operating mode. Let us consider, for example, the loading of the batch FORTRAN compiler, assuming that the user has already constructed a file of FORTRAN source code.

Upon receiving a FORTRAN command, the SYS1 module is loaded. SYS1 safe-stores the contents of the GUTS module containing critical global variables and compares a list of core and peripheral file storage at hand with that needed by the compiler. SYS1 requests additional core and file space as needed from the GECOS monitor. The FORTRAN compiler, the GMAP assembler and a few other programs loaded by GUTS are considered 'well behaved'. That is, they will run to completion of the assigned task and will return control to a specific core location upon termination. This being the case, GUTS flags FORTRAN accordingly and inserts a small boot-strap loader at the latter location. FORTRAN is then loaded and compilation is underway.

Upon completion of compilation, the boot-strap loader is invoked. This in turn loads module SYS2 which returns the borrowed core to the GECOS system, reloads the critical GUTS module and copies any required diagnostics and/or output into the internal scratch files. The TTY module is loaded, the user is informed of return to system command level and interactive computing then proceeds.

In some instances 'ill-behaved' programs are loaded. These are programs which request termination of the job, attempt to vary their own core allocation or are likely to abort due to software error. A RUN command, in which a user attempts to load and execute his own program, usually possesses at least one, if not all, of the above characteristics. In this case, the SYS1 module loads SYS3 which in turn runs the user's program in a protected mode, trapping those operations which might be detrimental to GUTS recovery. The cost paid by the user for this mode of operation is an overhead in processor time of approximately 10 to 15% and an additional 1024 words of memory occupancy in which SYS3 resides. Fatal errors are trapped by SYS3 which issues appropriate diagnostics to the teletype and loads SYS2 which in turn releases core and the process continues as in the previous case.

System performance

Operation

Let us again consider the GUTS memory map of Fig. 1. We have already noted that the ordinate represents system function. If along the axis of the abscissa we represent another parameter of interest, for example, processor time, we obtain some insight into the system operation. Block A of **Fig. 2** represents the loading and operation of module FTF for 100 milliseconds. This is followed by the loading of modules FTF1 (Block B) for 200 msec. and FTF2 (Block C) for 300 msec. By examining the

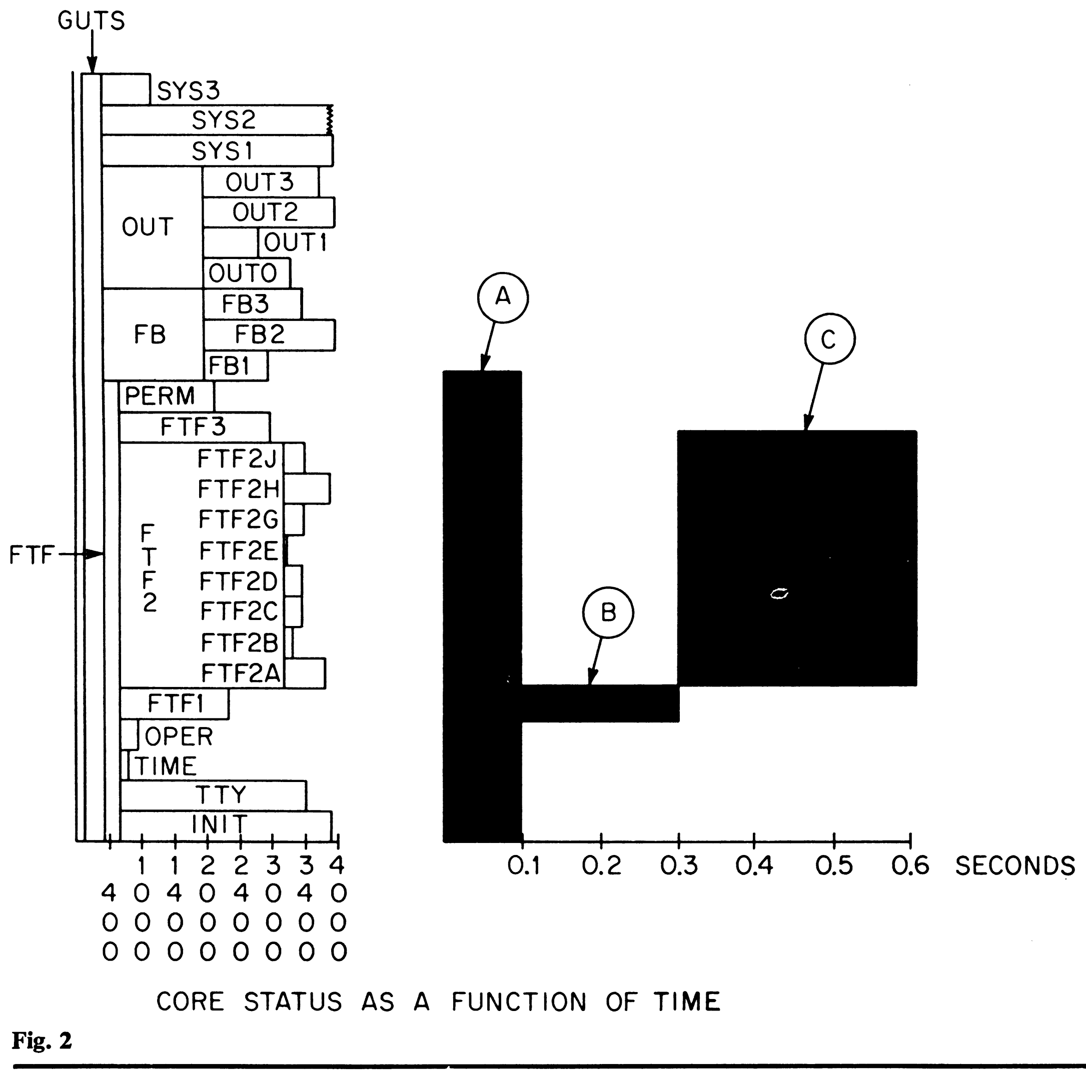


Fig. 2

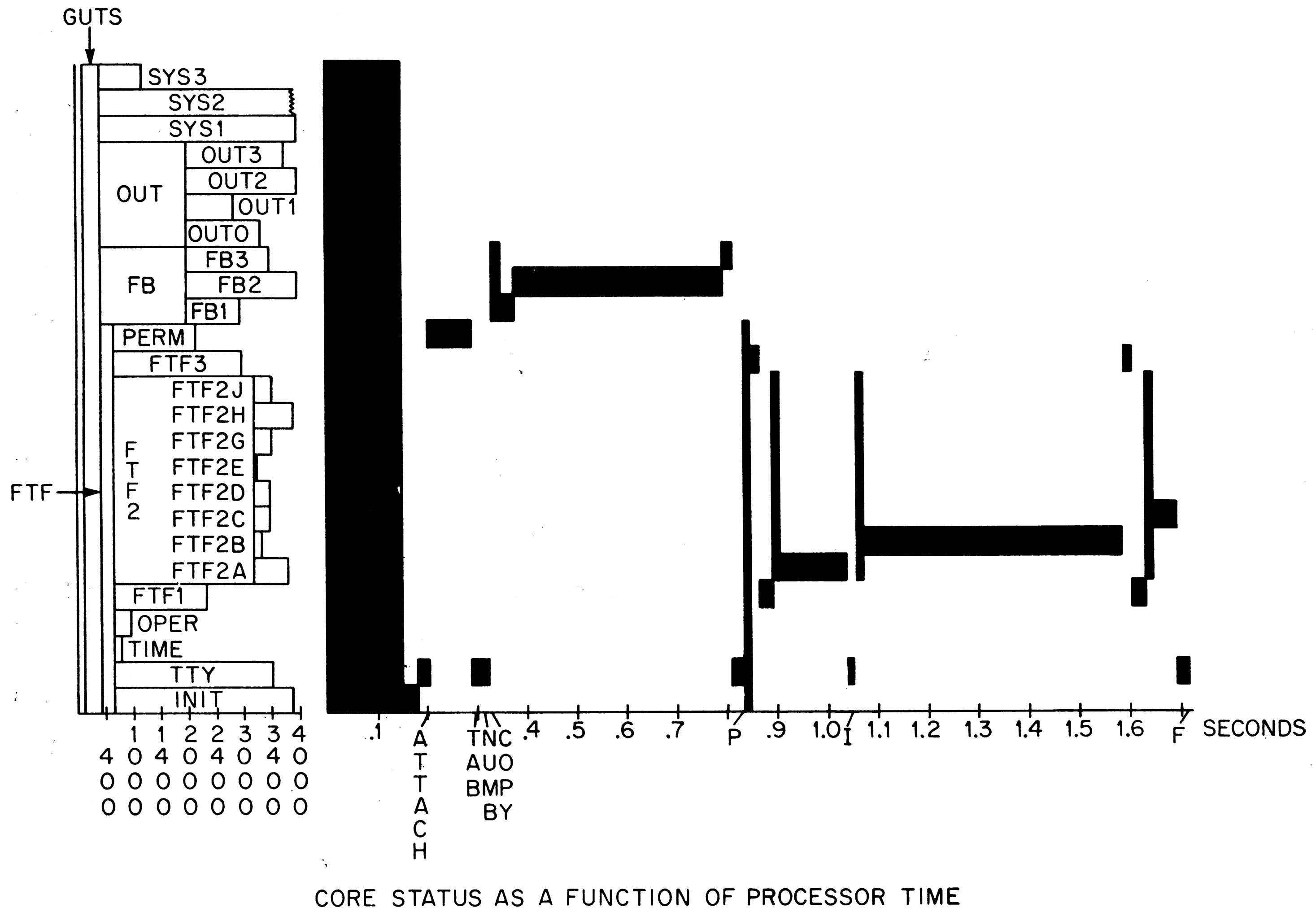


Fig. 3

occupancy requirements of these modules, one sees that FTF1 and FTF2 can use common data and subroutines resident in FTF.

Fig. 3 shows a portion of an actual session. As in the example of Fig. 2, processor time is the parameter being examined. The performance characteristics shown in this and successive figures represent typical sessions with many other jobs simultaneously undergoing execution in the GECOS multiprogramming environment, control being dispatched to GUTS in normal 'round-robin' fashion.

As shown in Fig. 3, the initialisation and accounting operation consumes the first 180 msec. of GUTS processor time. Control then passes to the module TTY which processes user command-level requests. The first user command (ATTACH) requests access to a file on semipermanent disk. Module PERM is loaded, 100 msec. is consumed, TTY is reloaded, and control returns to the user. Tabulation and line numbering modes are set. These commands are processed within TTY and require no other modules. The user then requests the copying of a file into the editing buffers. A buffer for the input file is loaded (FB), the file is opened (FB1), the file is copied (FB2) and closed (FB3). Control returns to TTY.

The print command ('P') forces the loading of a second internal editing buffer (FTF) and after some file manipulation results in the residence of module FTF2A which prints the appropriate line images on the teletype. The input command ('I'), loads FTF2B for file positioning and then loads FTF2C which reads text from the teletype.

The session shown in Fig. 3 is displayed again in **Fig. 4** where the abscissa is now elapsed (connect) time. Note the dominance of modules TTY, FTF2A and FTF2C corresponding to command level, output and input modes, respectively. **Figs. 5 to 7** show the operation of a complete session. The FORTRAN compiler was invoked twice and one assembly was performed. During this time, none of the GUTS modules shown on the left are present. By defining memory residence as the product of memory occupancy and processor time, the pictorial technique described above can be used to show the relative dominance of interactive and non-interactive modes. Fig. 7 displays memory residence as defined above adjusted to exclude the effect of the software which measures GUTS performance. The effect of this software on elapsed and processor time measurements is negligible. The memory residence shown in this figure for the two compilations and the assembly is identical to that which would be incurred by batch submittal of the identical tasks.

Measurement

A preceding section dealing with user feedback dealt with the need for determining who was using the system. In order to achieve a greater understanding of system performance and operation it was necessary to obtain measurements on how GUTS was used. The loading of every module was accompanied by the generation of a record of module name, processor time, elapsed time, input-output time and memory residence. During transitions between interactive and non-interactive modes a record was kept of the size of the requested core, the number of tries required and the type of processing being performed when core was allocated. Some of the results of measurements taken for 60 hours of user connect time are shown in **Tables 2 and 3.**

As can be seen from these tables 80% of the user holding

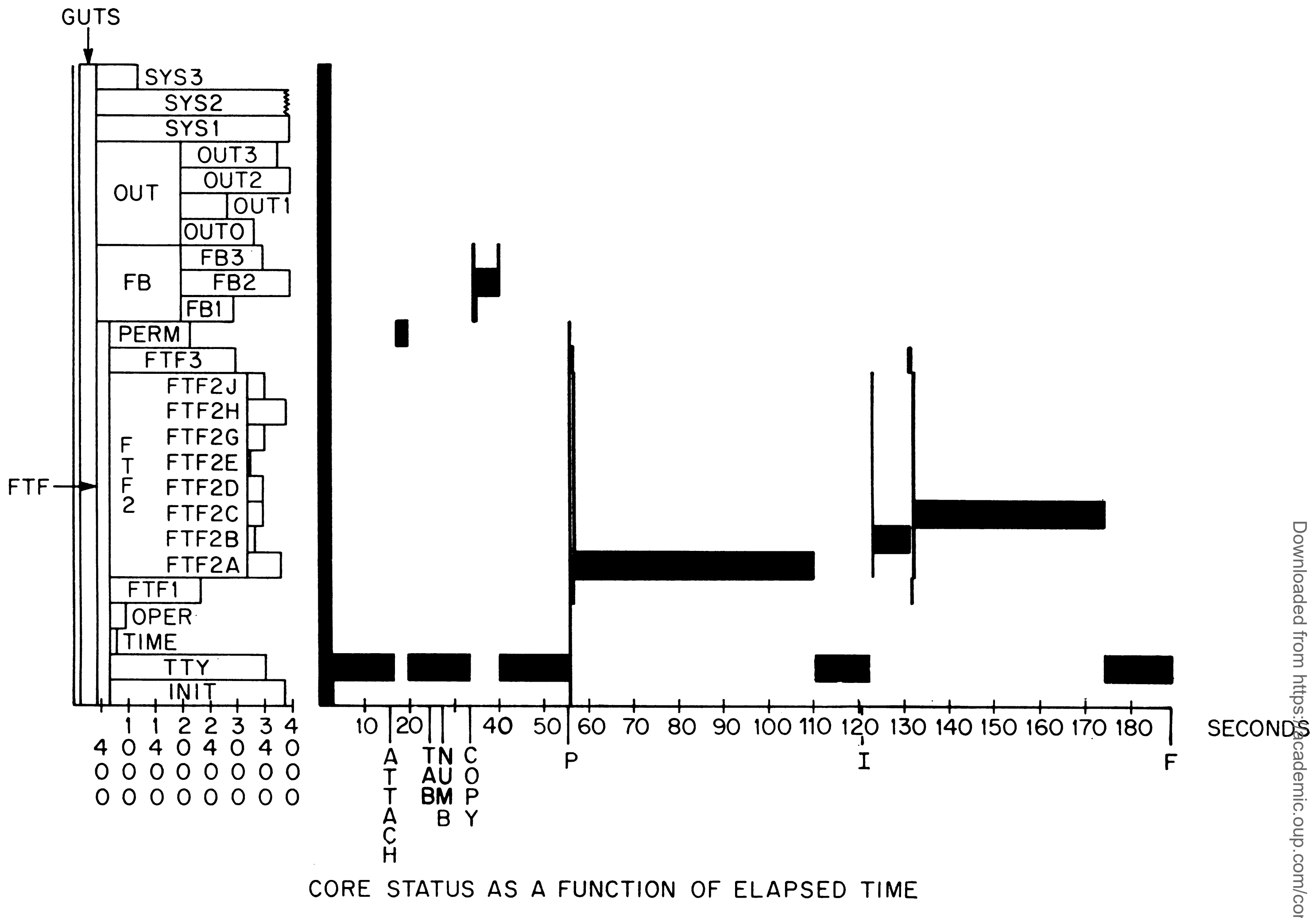


Fig. 4

(connect) time was spent in edit, input or output mode (TTY, FTF2A, FTF2C). However, these modules accounted for 15% of the accumulated GUTS processor time. Of the total accumulated connect time of 60·44 hours, 56·93 hours or 94·2% was spent in interactive mode. This compares with accumulated processor time of 0·904 hours with 48% in interactive mode.

Two hundred and fifty requests were made to the monitor system for more core. The average core request was for 20·6K and was fulfilled after 3·25 tries requiring a wait of 52·05 seconds. Once obtained this core was held for 47·97 seconds. Mean core occupancy excluding the overhead imposed by the measuring module was 4,100 words. This compares favourably with the present per-user core occupancy of GE-TSS. The load imposed by the user on the Input Output Controller (IOC) due to the loading of modules was an average of 11·92 seconds per session with a mean of 17·45 seconds of IOC time for internal editing scratch files.

The waiting time imposed by GUTS due to loading of successive modules was of much concern during early stages of design of the system. A typical worst case situation would arise during the transition between the copying of an external file and the printing of its first line. Following the print command, it is necessary to close the present scratch file (load FTF3), load a second scratch file buffer (FTF), open both files (FTF1), load the I/O software (FTF2) and print module (FTF2A). Typical response time for the above (excluding the

Table 2 Module residence time (Core occupancy for modules not shown less than 1·2% of total connect time)

GUTS	94·2%
TTY	44·7%
FTF	83·2%
FTF2	37·8%
FTF2C	19·4%
FTF2A	16·1%
SYS1	6·9%
Non-Interactive Mode*	5·8%
OUT	1·6%
FB	1·2%

*Includes modules SYS2, SYS3, compilers, assemblers, utility and user programs.

Table 3 Module processor time (Modules not shown used less than 1·52% of GUTS processor time)

Non-Interactive Mode*	52·3 %
TTY	7·15%
FTF2A	7·06%
OUT2	5·43%
FTF2B	4·35%
FTF2J	4·34%
FB2	4·26%
PERM	1·93%
FTF2	1·62%
SYS1	1·52%

*Includes modules SYS2, SYS3, compilers, assemblers, utility and user programs.

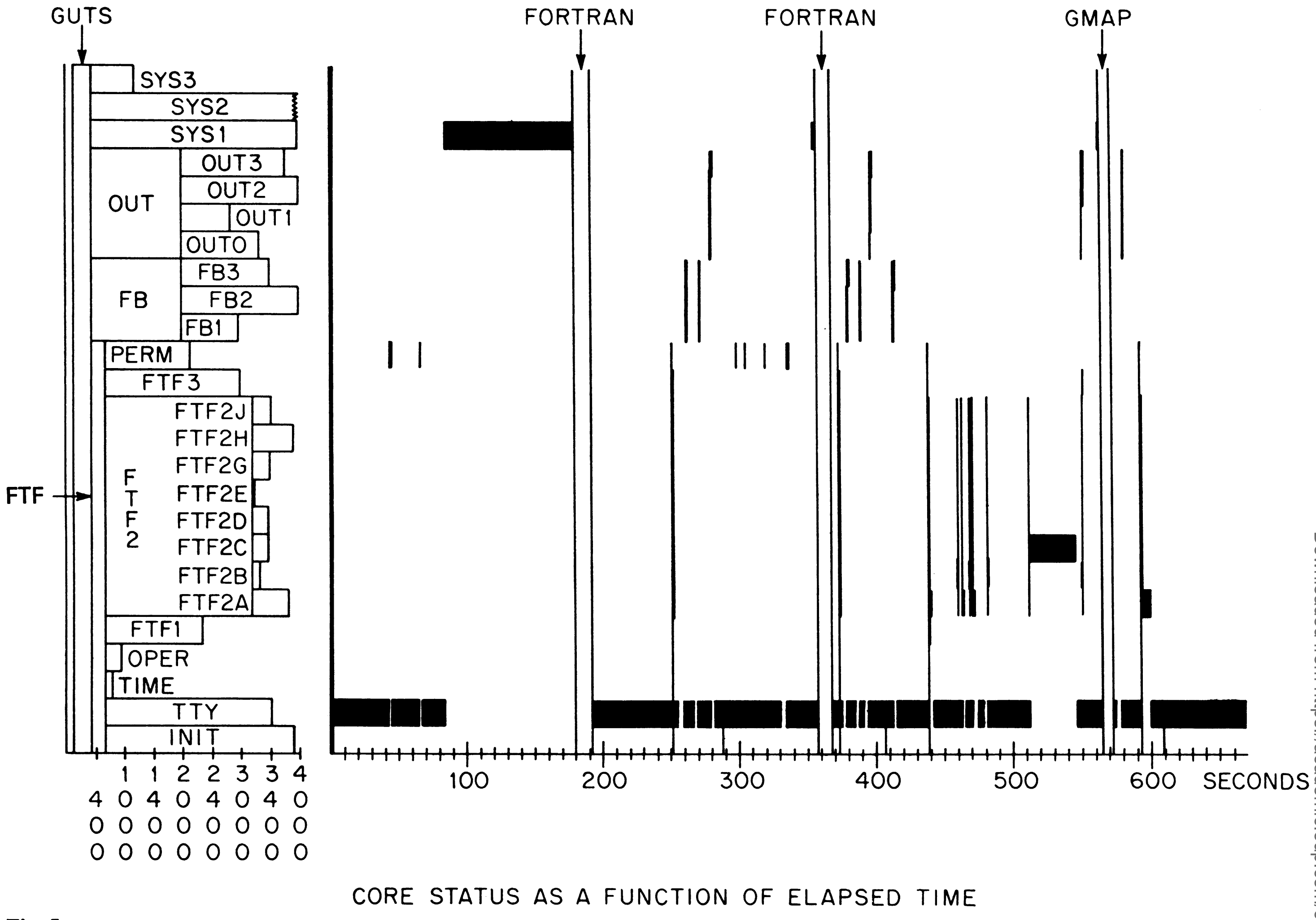


Fig. 5

time for the actual transmission and printing of the line), during prime shift on a loaded processor was approximately 1·2 seconds. When already in editing mode, the response time to a second editing command involved loading only the I/O software (FTF2) and the appropriate functional module (FTF2A, FTF2B, FTF2C, etc.) resulting in an average pause of 0·3 seconds in cases where extensive file positioning was not required. This response was considered to be satisfactory.

An examination of the system delay due to requests for core associated with the loading of the FORTRAN compiler or the GMAP assembler yields results which are at best 'tolerable'. The average assembly or compilation required 25 seconds of elapsed time for execution while imposing an additional delay of 44 seconds for core allocation.

Interaction

The relatively low cost of interactive operation displayed in Fig. 7 is characteristic of system performance. However, it should be noted that both the FORTRAN compiler and the GMAP assembler present only limited interactive facilities. In both cases, the source input code is completely specified before compilation or assembly, and diagnostics were presented to the user after completion of the process.

A more serious limitation is the difficulty in executing interactive code. The interactive modules which were loaded by user commands were highly tailored for reduced memory residence. Although GUTS possesses no restriction upon the execution of user's interactive code*, it has no ability to deal effectively with the problem of memory residence of a user program. Thus, the execution of a user's code is effectively restricted to limited interactive operation as in the case of compilation.

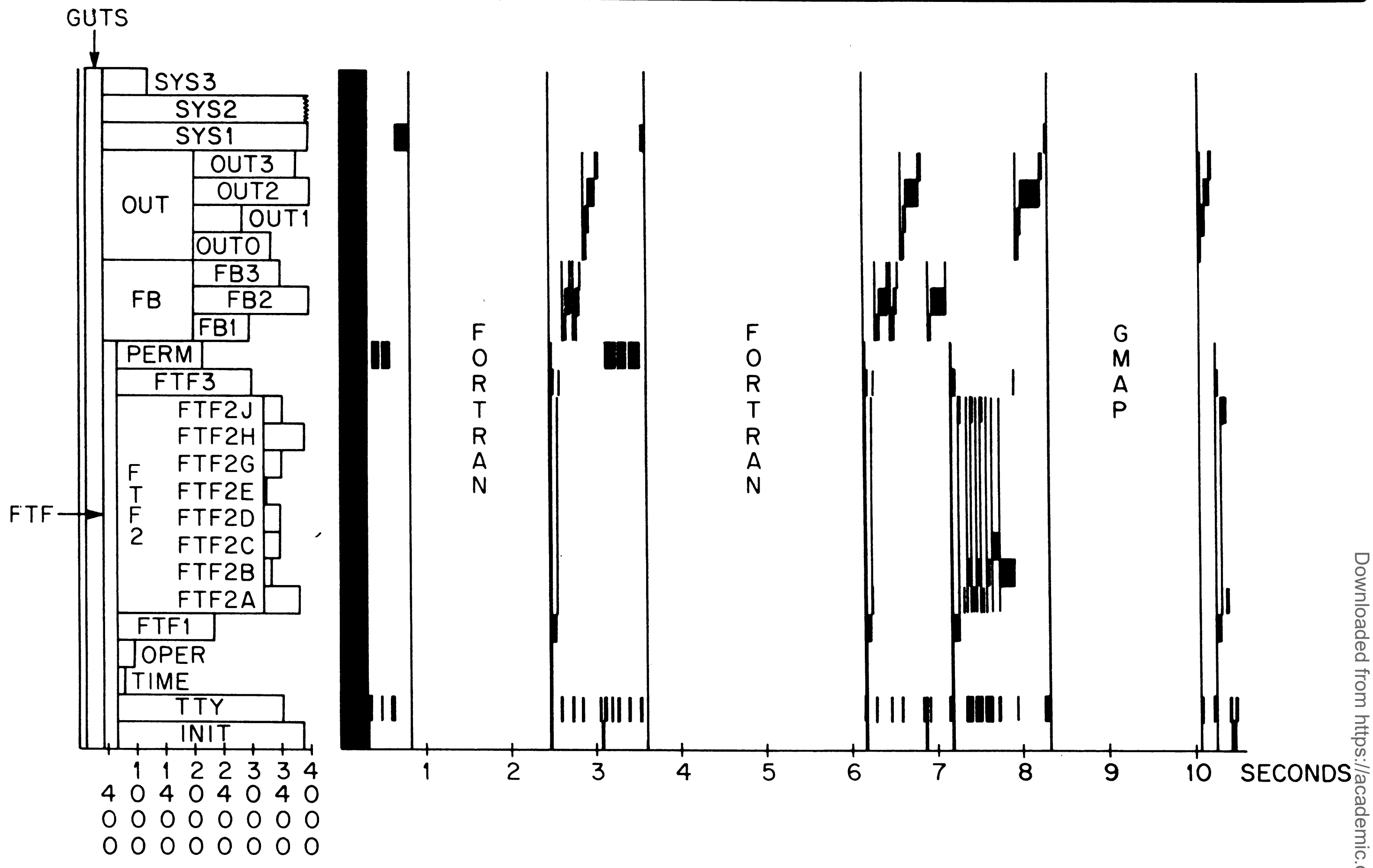
A possible solution to the above limitation would be to design the protected mode supervisor (SYS3) to swap a user's program to secondary storage and reduce core occupancy while waiting for teletypewriter response. Although theoretically possible and easily implemented within the framework of GUTS, this approach would be impractical due to the time required by GECOS to fulfill a request for additional core upon conclusion of each teletype interaction. Although the typical delayed response for core acquisition was considered tolerable when compiling, assembling or initiating execution, a similar delay during question and answer interaction would present a very poor human engineering interface.

At present, the GECOS monitor processes requests for additional core allocation on an 'as-available' basis. That is, additional memory occupancy is granted if, and only if, the requested storage is unoccupied, GUTS depends heavily on the core dynamics of its host GECOS system. It seems probable that improved interactive response would be obtained if the multiprogramming core allocation discipline was conducted on a priority allocation basis, interactive job requests in some sense having higher urgency than jobs for batch submission. No data is presently available dealing with expected response to core allocation based on this priority service, nor has the author any information on the cost, in terms of system overhead, that would result.

Conclusions

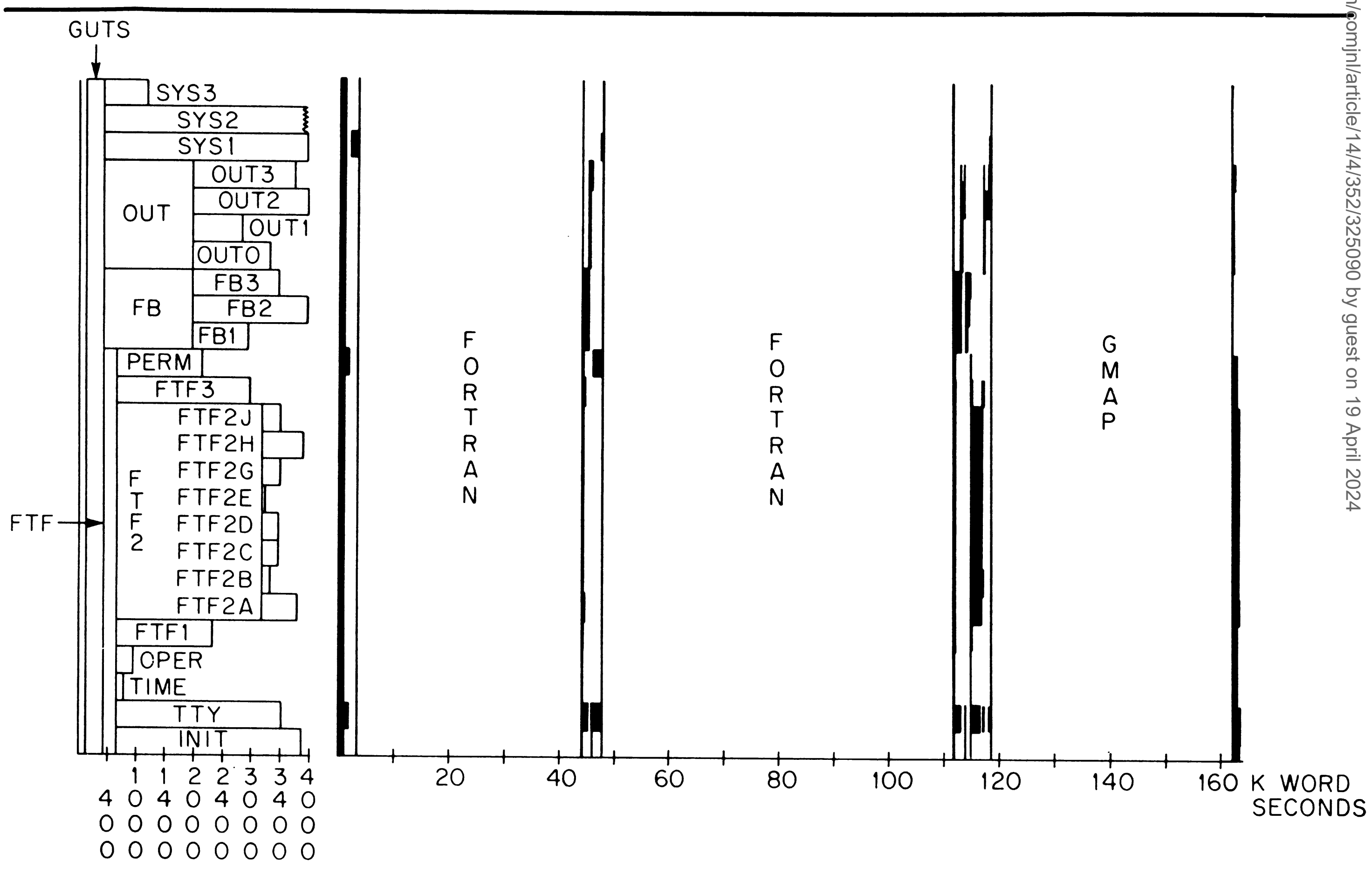
The preceding pages were devoted to a discussion of some of

*It is of interest to observe that a GUTS user can run any program including a second copy of GUTS.



CORE STATUS AS A FUNCTION OF PROCESSOR TIME

Fig. 6



MEMORY RESIDENCE

Fig. 7

the design objectives and performance characteristics of an interactive batch-compatible computing facility. GUTS was added to, in a sense on top of, an already existing batch operating multiprogramming system. Approximately 1½ man-years of effort have been devoted to its design and implementation. Less than 1 man-month of this effort was spent on developing a preliminary version which was used, on an interactive basis, to develop successive refinements.

GUTS features many facilities that enable a user to deal effectively with large programs. The ability to divert output from teletype to high speed printer, card punch, microfilm, or on-line electrostatic printer, has been an invaluable asset both in the development of this system and in the undertakings of other users. The incorporation of extensive software systems developed by others into GUTS commands is a relatively painless process, usually requiring no change to the incorporated software and the insertion of a few command defining macro-instructions in GUTS.

Three approaches for providing an interactive computing environment were enumerated in the Introduction to this paper. Our discussion on the limitations of this interaction led us to conclude that the execution of programs requiring interactive dialogue might be practical if some changes were made to the multiprogramming supervisor. The costs involved and the compromises that would result seem to be fruitful areas for further study. It appears then, that there is yet a fourth approach to interactive batch-compatible computing which would maintain the user isolation features of GUTS while recognising the priority requests of interactive dialogue programs on the multiprogramming supervisor level.

The above paper has outlined an approach to providing an interactive batch-compatible computing facility. The system has proven to be easily implemented, easily maintained and economical (in terms of measured system performance) to operate.

Reference

GE-600 Line GECOS III Time-Sharing System, System-Programmer's Reference Manual, CPB 1514B, General Electric Company, Phoenix, Arizona.

Correspondence

To the Editor
The Computer Journal

Sir,

Your editorial in the February 1971 issue of *The Computer Journal* (page 1) made me read the issue with your comment in mind about 'the feeling among members that only a limited section of each issue is of any interest to them'.

As usual there is very little in the February issue about business applications, which is the common cause of complaint. I see very little hope of changing this. The academic and the researcher have as a major objective of their professions the publication of their work. The worker in business is not judged by what he publishes. He is inhibited from publishing by two time-honoured excuses—lack of time, and commercial security. Short of paying for papers from business users I do not see what the Society can do.

I can be more constructive on another major issue, that is the clarity of the papers that do appear. The February *Journal* contains much that is instructive.

First there is Brigadier Allen's letter (page 86) on the use of English. Despite your editorial, we do not look for Shakespeares. A better model is the terse style of *Which?* Your referees and your Editorial Board can do a great deal. The bugbears are the use of jargon, and of long sentences and paragraphs where even the writer loses the thread. Fine examples of both can be found in February.

Next there is the problem of symbolic notations. This is the greatest barrier between the theoretical and the practical man. The researcher in a new and specialised topic needs to invent a new symbolism in order to progress. But this itself cuts him off.

The case is not hopeless. I know that an ordinary programmer can learn new symbols if he must. Backus's notation looked daunting to me when I first saw it. Like others I mastered it when I had to and it is now common currency. Or is it? How many *Journal* readers pass over the papers which use it, simply because they have not been offered a plain explanation of it?

So the question arises, should the *Journal* be the means for the specialists to talk to each other, or their means to talk to the average member? If the latter, then they have to teach us their new symbols, in detail and with patience. It can be done—I point to the February *Journal* and to Professor Michie's paper (page 96) on 'Heuristic Search'.

Professor Michie says that his tutorial 'gives an introduction to the theory and surveys the state of the art'. There is a great need for this sort of thing. Professor Michie's group contributes another good example (page 91) with its 'Tokyo-Edinburgh dialogue'. Of course they do work on a subject whose interest is compelling even to those who foresee no direct use for it—even so, from the start of their work their efforts to let us know about it have been a model for all researchers.

The *Journal* is not the only medium for spreading new ideas, but at present it is the best we have. P. J. H. King refers in his letter (February, page 54) to a 1969 ICL manual 'taking the current state of development of a topic as exemplified by current papers in the literature and the discussions amongst those interested in the field, and reducing them to a set of recommendations for practical day-to-day use . . . it is clear that this type of activity will become increasingly important.'

Well, yes. But my company is a big ICL user, and the *Journal* mention is the first I have heard of the manual concerned.

Perhaps I have got it all wrong. You may think I expect more of the *Journal* than it can do. In that case, let it remain more or less the learned publication it has been, and let the Society find other ways of fulfilling its duty to broadcast new ideas to computer users.

After all, does not that duty extend beyond a duty to registered members of the Society? There are many installations where no staff, or no senior staff, are in the Society.

A fresh approach could be to bring out a new monthly publication, possibly in partnership with a commercial publisher. I have in mind the way the British Institute of Management participates in *Management Today*. However *Computing Today* would concentrate on promulgating the techniques and achievements of advanced workers in both research and business to the mass of users. It would be sold or subscribed for, and would pay its contributors.

If three publications are too many, then consider whether *Computing Today* might also replace *The Computer Bulletin*. There would be room in it for what is important in the *Bulletin*.

There may be other choices. I hope this letter helps to get them examined by Council.

Yours faithfully,
J. W. Gale

16 Wrensfield
Hemel Hempstead
Hertfordshire
24 April 1971